A Connection of Task-centric with Artefact-centric Models through Semantic Task Specification and its Use for Formal Verification

Roman Popp, Ralph Hoch, and Hermann Kaindl
TU Wien, Institute of Computer Technology, Vienna, Austria
{roman.popp, ralph.hoch, hermann.kaindl}@tuwien.ac.at

Abstract

Task- and artefact-centric business process models (BPMs) are mostly used in isolation. This entails, e.g., problems with formal and automated verification of BPMs through model checking. *We address this gap through* semantic task specification*, which is transferred from more widely known semantic service specification. In summary, we present a new and systematic approach for connecting a task-centric BPM (in BPMN) with a model of an artefact-centric object life cycle through semantic task specification. As a consequence, we achieve a seamless approach for formal and automated verification of BPMs using model checking.*

1 Introduction

While task-centric models (of business processes) focus on the overall behavior (of the process), artefact-centric models represent the state changes of each artefact. In this sense, they are complementary, and they are mostly used in isolation.

In practice, business process models (BPMs) are often represented in BPMN (Business Process Model Notation) [12] in a task-centric way, where the focus is on control flow. Their properties, however, often relate to the states of objects involved, which can be defined in an artefact-centric way using *object life cycles*. Unfortunately, BPMN models are usually not connected with models of object life cycles, and the effects of their tasks on business objects are not defined.

We propose a connection of such models and its use for formal verification of BPMs. This connection is by means of *semantic task specification*, inspired by annotations of activities with logical preconditions and effects according to [16], which have been inspired in turn by semantic Web service approaches. Metaphorically speaking, we utilize the additional semantic knowledge on top of the control flow of a BPMN-like BPM to connect it with a semantically specified object life cycle.

According to a recently introduced and promising approach to formal verification of BPMs given in BPMN based on *model checking*, formally represented properties refer to additional models of *object life cycles* [7]. In this way, it is actually possible to formulate properties to be checked without knowing the BPM model that is to be verified. This approach even allows the verification of several BPMs connected with the related object life cycle(s) against the same set of properties. Unfortunately, the connection of models is established based on informal annotations of BPMN models only. Hence, establishing the required connection of models in a rigid way has not yet been solved. Actually, also the way of transforming BPMN tasks as proposed in [7] is not fully consistent with the representation of object life cycles as used for model checking.

We build on this approach, but make it rigid for solving these problems, and in order to facilitate its automation in a seamless formal verification process. In fact, we use our new connection of task-centric with artefact-centric models through semantic task specification for this purpose.

The remainder of this paper is organized in the following manner. First, we present some background material, in order to make this paper self-contained, and relate it to previous work. Then we present our new and rigid approach for connecting task- and artefact-centric models logically through semantic task specification. Based on that, we explain how to connect also their behavior for formal verification through model checking. Finally, we discuss our approach more generally and conclude.

2 Background

2.1 Background on Semantic Service Specification

Semantic service specification has been based upon the Web Ontology Language (OWL), which is a knowledge representation language used to build and administer on-

URI: http://hdl.handle.net/10125/41748
ISBN: 978-0-9981331-0-2

tologies or a specific knowledge base.[1] OWL-S [1] is an ontology built upon OWL for semantic descriptions of Web services. OWL-S consists of three parts: *Service Profile*, *Process Model*, and *Service Grounding*. The latter provides the means for interoperability with a Web Service Description (given in WSDL) and relates the semantic specification of a Web service with its WSDL file. This involves the definition of the input and output parameters including their types. In addition, OWL-S provides pre-defined predicates for defining preconditions, result values and effects. Services can be modeled either as atomic or as more complex composite services, where the latter consist of several (orchestrated) atomic services.

To illustrate how semantic specification works, let us consider the task Pay Invoice as an example. Within this task, an *Invoice* is to be paid, and after that passed along according to the control flow in the BPMN model. The task itself is specified through its input and output, but it lacks a semantic specification that describes what the task accomplishes. The output alone is insufficient, as it only specifies the result of the task in the form of a type (in this case, an Invoice). However, it is not specified what kind of changes occur during task execution and how the domain in which the business process is enacted, is affected. This additional specification can be provided using the OWL-S formalism.

The *hasResult* predicate of OWL-S specifies the result of a service, where it couples both outputs and effects. Outputs are passed along from the service and correspond to an output variable from, for example, a WSDL file specifying a Web service. In addition, effects specify how the domain changes. To be more precise, they specify the changes that are caused by the service execution. Effects are specified with the *hasEffect* predicate. A semi-formal specification for the *Pay Invoice* Task (as inspired by [6]) is presented in Listing 1.

Pay Invoice:

Input:	Invoice
Output:	Invoice
Precondition:	authorized(Invoice)
Effect:	paid(Invoice)

Listing 1: Semi-formal *Pay Invoice* Specification

This task operates on an Invoice, which is passed to it as an input. On this input, a formal condition (precondition) is specified, which has to be fulfilled before the task can be executed. Here the precondition states, that the Invoice has to be authorized before a payment. The changes in the domain are modeled as effects and relate to the *hasEffect* predicate. In this example, they specify that the Invoice has been paid after the task has been executed. The predicates *authorized* and *paid* are concepts of a domain ontology and can be used in combination with a rule language such as SWRL [15].

[1] Web Ontology Language: http://www.w3.org/TR/owl2-overview/

Each SWRL-Rule consists of a head and a body. The head is deduced if the conditions in the body evaluate to true. So, SWRL-Rules provide the means to deduce knowledge from existing facts. A typical example of an SWRL-Rule is the following relationship in families: if a parent of a child has a brother, than it can be deduced that this child has an uncle. Listing 2 shows how such a rule can be defined.

```
hasParent(?child, ?parent) ∧
    hasBrother(?parent, ?brother) ⇒
    hasUncle(?child, ?brother)
```

Listing 2: Example of an SWRL-Rule

This notation is used since all the basic knowledge in OWL is specified in the form of triples. They consist of a *Subject*, a *Predicate* and an *Object*, where a predicate relates a subject to an object. The notation *hasParent(?child, ?parent)* states that the *?child* is in a relationship with a *?parent* through the predicate *hasParent*. In this example, the *hasParent* predicate is used to check if two individuals *?child* and *?parent* are related. They are only related if a triple of the form *?child :hasParent ?parent* exists in the knowledge base. The rule is checked for all available individuals.

There are several possibilities available to specify such SWRL-Rules. For example, they can operate on classes or on individuals (instances). One possible OWL representation of the rule above is shown in Listing 3. In this case, the rule operates on concrete individuals. These are identified via the *child* and *parent* variables, and all available instances in the domain are used.

```
<swrlx:individualPropertyAtom
    swrlx:property="hasParent">
  <ruleml:var>child</ruleml:var>
  <ruleml:var>parent</ruleml:var>
</swrlx:individualPropertyAtom>
```

Listing 3: Excerpt from SWRL-Rule Example

Such rules are often used to describe effects in OWL-S, as they show how the state of the domain changes. More precisely, the effects specify how the previous state of the domain is transferred to the new state after task execution. In essence, they allow the deduction of new knowledge based on the task execution.

2.2 Background on Model Checking Using Object Life Cycles

Model checking (or property checking) is a formal verification technique based on models of system behavior and properties, specified unambiguously in formal languages (see, e.g., [2]). The behavioral model of the system under verification is often specified using a Finite State Machine (FSM), or several synchronized FSMs, more precisely asynchronous FSMs synchronized by signals. The properties to be checked on the behavioral model are formulated

in a specific property specification language. When a model checker finds a property violation, it reports it in the form of a counterexample.

This general verification approach was adopted for verification against business rules using *object life cycles* in [7]. The primary motivation for employing additional models of object life cycles was to allow formalizing properties (according to business rules) without knowing the process model. However, these models have to be connected with the BPM to be checked in the formalism used for model checking, and this was only done based on informal annotations of BPMN models.

Instead of them, we define *semantic task specification*. This facilitates a rigid and seamless verification process based on model checking and object life cycles.

In order to facilitate the comparison of our work with the one in [7], we use the same running example as shown in Figure 1. This process model given in BPMN is based on a simplified version of the "payment handling" process of [14, p. 108], which is used here as a reference process. This payment handling process starts with the creation of an Invoice. This Invoice is then sent to the Customer. After the Customer has received the Invoice, she makes the payment of the Invoice. Once the payment is received, it is booked as paid. After that, this example reference process is finished. While the payment is unconditional in this reference process, the one in Figure 1 actually includes a conditional authorization.

We renamed the tasks Send Invoice to Transmit Invoice, and Receive Payment to Receive Paid Invoice, in order to achieve a cleaner terminology. Still these BPMs per se are isomorphic. Instead of informal annotations, we added Pre and/or Eff statements, as inspired by [16]. In our approach, these illustrate semantic task specification as explained below.

3 Related Work

Given a formal representation in a BPM, checking correctness properties inherent in the business process itself is possible. Wynn et al. [17] verify business processes against four defined properties (soundness, weak soundness, irreducible cancellation regions and immutable OR-joins). Sbai et al. [13] show how a model checker can be used to identify problems with a specification of a business process to be automated as a workflow, and how a verification of certain correctness properties can be accomplished.

Some previous work addressed the question of what to verify a business process model against, to determine possible violations of certain properties given in addition to the process model itself. Fisteus et al. [5] propose a framework for integrating BPEL4WS and the SPIN and SMV verification tools. This framework can verify a process specification against properties such as invariants and goals through model checking.

Lohmann et al. [8] present an approach based on compliance rules, which are used to automatically construct artefact-centric BPMs that are compliant by design. The building blocks are life cycles of the involved artefacts, which are used to generate compliant BPMs. The compliance rules express constraints on these artefacts as well as the actions performed on them. With this approach they are able to reduce the check for compliant behavior to the reachability of final states.

Meyer et al. [10] define a "weak conformance" between process models and synchronized object life cycles. Their algorithm for soundness checking verifies whether each time an activity needs to access a data object in a particular state, it is guaranteed that the data object is in or can reach the expected state. In contrast to our approach, they do not verify against additionally specified properties.

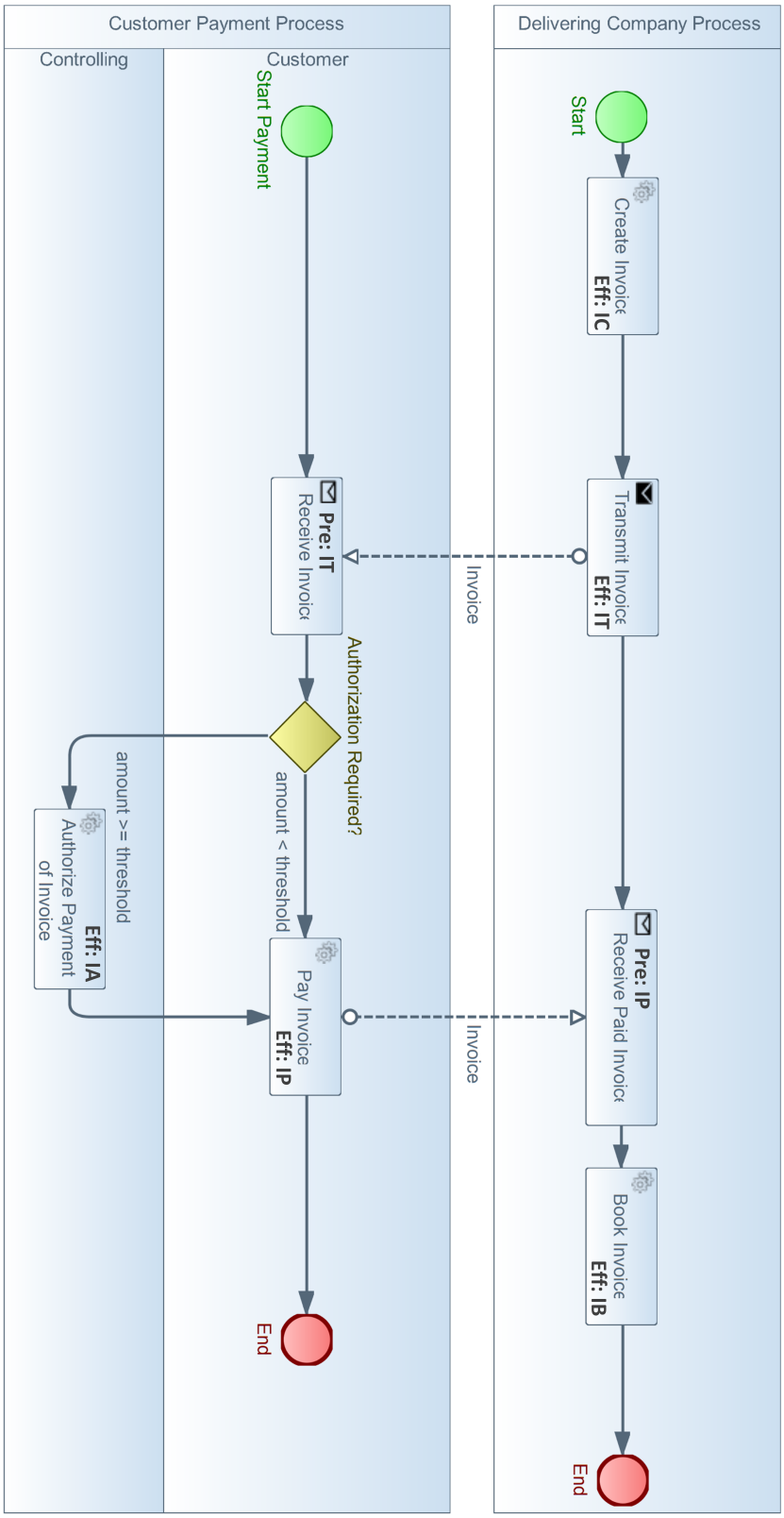
Estãnol et al. [3] propose a verification approach based on artefact life cycles modeled in UML. It checks certain intrinsic properties such as liveliness of a class or an association.

Feng et al. [4] propose an approach for verifying properties of an OWL-S service process model. Via mapping rules this approach translates the process model into a process algebra model and uses a model checker to verify the properties of such a translated model. It handles the control flow as well as the binding-based data flow of the process model. In contrast to our approach, there are no models of object life cycles involved.

Ni et al. [11] transformed models and formally verified semantic Web services composition. More precisely, their approach verifies the correctness of semantic Web services composition based on models of Colored Petri Nets that are transformed from OWL-S models. It is sound to use such Petri Nets in order to verify reachability and soundness of composed services (among others). This approach differs from ours as it does not address the connection of process models with object life cycle models.

In the context of verification of business process models, Weber et al. [16] addressed the problem that control flow does not capture what the process activities actually do when they are executed. So, they annotated individual activities with logical preconditions and effects, specified relative to an ontology with axioms of the underlying business domain. This allowed them to verify the overall process behavior, but they do not use semantic task specification for model checking as our approach.

A recently introduced approach for automated verification of business processes using model checking, shows how BPMs can be verified against properties defined on object life cycles [7], see also the background material above. This approach is promising, but the connection of the BPM



IC = InvoiceCreated; IT = InvoiceTransmitted; IA = InvoiceAuthorized; IP = InvoicePaid; IB = InvoiceBooked.

Figure 1: Payment Handling Process in BPMN, Including Authorization

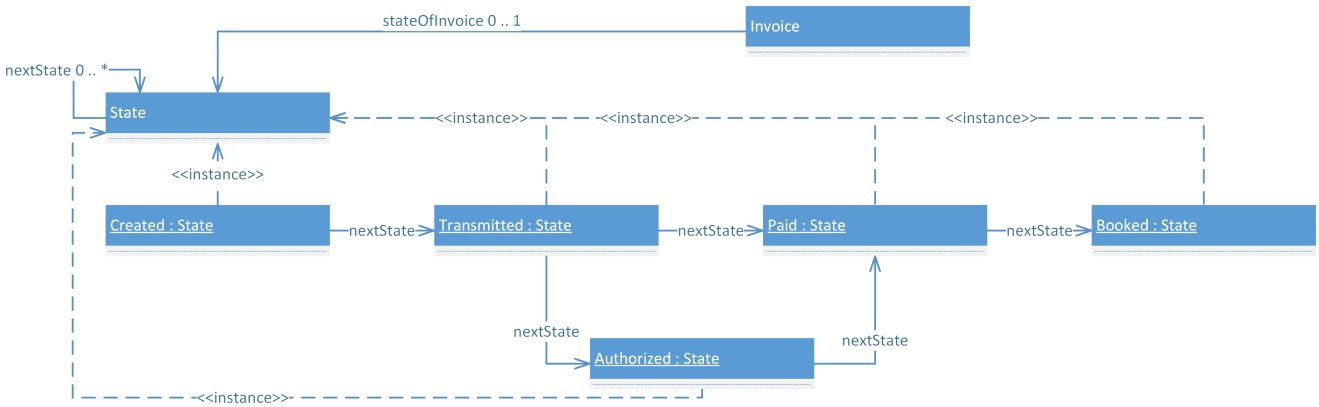


Figure 2: Part of an Ontology of Domain Objects and Their Life Cycles

and object life cycles is established based on informal annotations given in the BPMs. In addition, they translate BPMN Tasks to states in the FSMs and thus mix conceptual views. Tasks in BPMN have a behavior and change the state of the domain in which they are enacted as opposed to states in FSMs, which represent a specific state of the process. We build on this approach, but make it rigid for solving the problem of connecting the models, and in order to facilitate its automation. Furthermore, we fix the problem of defining tasks as states in the resulting FSMs.

In summary, we are not aware of any previous work that studied semantic task specification for a systematic transformation to synchronized FSMs and a rigid connection of task-centric BPMs with artefact-centric object life cycles as presented in this paper for a seamless approach to formal verification of BPMs.

4 Connecting Task- and Artefact-centric Models Logically

For logically connecting task- and artefact-centric models, we adopt parts of OWL-S specifications previously defined for services and employ them for semantic task specification. In essence, we utilize the possibility to specify conditions and effects, to provide a formal semantic specification of all tasks that are embodied in a BPM. That is, each task in a BPM has a counter-part in OWL-S, which semantically specifies its behavior. The conditions and effects relate to the object life cycle defined in OWL and thus connect the two models together.

4.1 Modeling Object Life Cycles in OWL

The verification approach given in [7] uses object life cycles to specify properties. Thus it is necessary to have a formal specification of these object life cycles. The concept of an object life cycle is independent of the domain that it is used in, and the concepts from the domain need to provide their own instantiation of an object life cycle. That is, each domain concept has its own definition of its life cycle by creating instances of the object life cycle concepts.

For this purpose, we introduce an ontology to model life cycles in OWL. This ontology defines that an object life cycle consists of several *States* and *Transitions* among them. The *Transitions* are defined with the *nextState* relationship. Furthermore, the *nextState* relationship also specifies which sequences of *States* are allowed in an object life cycle. For each domain object, a concrete new instance of the object life cycle ontology is created. The definition of the domain can be provided in OWL as well or be part of an Enterprise Architecture.

Figure 2 shows how this life cycle ontology is used for an ontology of domain objects and their concrete life cycles. More precisely, the figure shows how the domain object *Invoice* can be enriched with an object life cycle. The concept *Invoice* has a relationship to the *State* concept, which defines that each instance of an *Invoice* is in exactly one state in the object life cycle at any given time. The object life cycle of *Invoice* is modeled through instances of the *State* concept and its *Transitions*. In this example, an *Invoice* is *created* first, then *transmitted* and then either *authorized* or *paid*. The example also illustrates how more than one successor state can be defined.

4.2 Connecting Task Specifications and Object Life Cycles

As mentioned in the Background section above, OWL-S provides the means to semantically specify services. We use these specifications to relate services to our object life cycle ontology defined above. In essence, the preconditions and effects of a service are related to states in an object life cycle.

The predicate *hasResult* in OWL-S couples outputs and effects. Effects are closely related to tasks, since they specify how the domain objects change, and are specified via the *hasEffect* predicate. To be more precise, they specify the changes that are caused by the service execution. In our approach, we relate these changes to state transitions in an object life cycle. That is, each service execution may change the state of a domain object.

SWRL-Rules are one possibility to model such a behavior. In our example, we use such rules to specify the new state of an *Invoice* after service execution. The body of our rules is always true, since there are no conditions on the effect. So, each execution of a service always has the same effect in our case. OWL-S provides the means to use conditions on effects and outputs as well, but for our purposes a simple unconditional effect is sufficient.

Let us consider our *Invoice* example again to illustrate this. A service *Create Invoice* is semantically specified and has the effect that an *Invoice* is created. In this case, the effect of the service execution is that the state of the *Invoice* is set to *Created*. Listing 4 shows this effect as an SWRL-Rule.

```
true ⇒ stateOfInvoice(Invoice, Created)
```

Listing 4: Effect of Create Invoice

The complete semantic specification of the task in OWL-S is very verbose. Due to lack of space, only an excerpt containing the output and the result along with its effect is shown here in Listing 5. The task is identified via the *rdf:about="CreateInvoice"* statement and is modeled as an *AtomicProcess*. An *AtomicProcess* executes an atomic operation, for example a WSDL operation. The next part, starting with the predicate *hasResult* in Listing 5, describes the result. It shows the output (*rdf:ID="InvoiceOutput"*) and its binding. The binding is actually omitted here, since it does not influence the specification of the effects. These are specified in the *hasEffect* predicate. In Listing 5, an SWRL-Rule is used to define the effect. The rule is identified by the rdf:ID *StateTransition*. Recalling that all statements in RDF are represented as triples of the form *Subject*, *Predicate* and *Object*, we can determine the meaning of the rule. The rule basically specifies that in our domain ontology the predicate *stateOfInvoice* is set for the subject *InvoiceOutput* to the object *Created*.

```
<process:AtomicProcess rdf:about="CreateInvoice">
  ...
  <process:hasResult>
    <process:Result>
     <process:hasResultVar>
       <process:ResultVar
            rdf:ID="InvoiceOutput">
        <process:parameterType
             rdf:resource="#Invoice"/>
       </process:ResultVar>
     </process:hasResultVar>
     <process:withOutput>
       <process:OutputBinding>
         ...
       </process:OutputBinding>
     </process:withOutput>
     <process:hasEffect>
       <expr:SWRL-Condition
            rdf:ID="StateTransition">
       <swrl:AtomList>
         <rdf:first>
           <swrl:IndividualPropertyAtom>
             <swrl:propertyPredicate
                  rdf:resource="stateOfInvoice"/>
             <swrl:argument1
                  rdf:resource="#InvoiceOutput"
                  />
             <swrl:argument2
                  rdf:resource="#Created" />
           </swrl:IndividualPropertyAtom>
         </rdf:first>
       </swrl:AtomList>
       </expr:SWRL-Condition>
     </process:hasEffect>
    </process:Result>
  </process:hasResult>
  ...
</process:AtomicProcess>
```

Listing 5: Excerpt of OWL-S Specification of Create Invoice Task with its Effect

With these specifications, the tasks are directly related to the object life cycles of the domain objects.

4.3 Connecting BPMN Tasks with Semantic Task Specification

In our approach, we use BPMN as the notation for BPMs. We connect each task of Figure 1 with a corresponding task specification in OWL-S. This is similar to the annotations in [16]. In case of existing task specifications in OWL-S, they can be reused, otherwise they have to be created from scratch.

For our systematic approach, each and every task in BPMN needs to have a task specification in OWL-S. Also for tasks that are not implemented by a Web service, a corresponding OWL-S specification has to exist. An example of such a task is *Transmit Invoice*. This task is a BPMN *Send-Task*, which is used to send messages to other tasks or processes. Such tasks do not use a service implementation for execution, but are directly executed by the BPMN engine. We employ OWL-S specifications for such tasks as well. Another example is the BPMN *Receive-Task*, which is the counterpart of a BPMN *Send-Task*. In contrast to the *Send-Task*, a *Receive-Task* waits for incoming messages.

BPMN defines an extension mechanism, which can be used to create custom extensions for BPMN elements. We utilize this extension mechanism to connect BPMN tasks with OWL-S specifications. The custom tag *serviceRef* defines the reference to the corresponding OWL-S service

specification for a task in BPMN. Listing 6 shows how the connection between BPMN and the semantic task specification is established.

```
<serviceTask id="CreateInvoice" name="Create
    Invoice" ...>
    ... // INPUT/OUTPUT Binding
    <extensionElements>
      <semService:serviceRef
          id="createInvoiceServiceRef">
        http://.../services#CreateInvoice
      </semService:serviceRef>
    </extensionElements>
    ...
</serviceTask>
```

Listing 6: Connect BPMN Task to Semantic Task Specification

5 Connecting the Behavior for Formal Verification

Based on the logical connection of task- and artefact-centric models, we are able to connect also their behaviors for formal verification through model checking. In addition to given FSM models of object life cycles, a prerequisite is a systematic translation from task models to synchronized FSMs. Using the resulting FSMs, the connection of the behaviors can be established based on the logical specification in OWL.

5.1 Transforming Artefact Life Cycles to FSMs

The model of the artefact life cycle shown in Figure 2 can be systematically translated into an FSM. Each state of the figure is translated to a state in the FSM. An exception is the first state, which is a modeled as a separate state and used as an entry point for the FSM. Figure 3 shows the result of the translation.

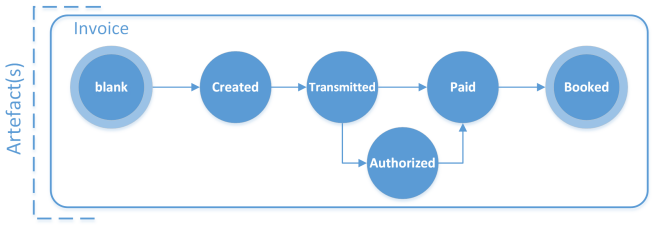


Figure 3: Object Life Cycle FSM

The FSM does not contain any signals among states. This is due to the fact that there are no signals yet available, and that the transitions are triggered from the process and not from the object life cycle itself.

5.2 Transforming BPMs in BPMN to FSMs

As shown in [7], formal verification of BPMs against object life cycles is possible. However, it is necessary to have a systematic transformation in place that translates BPMs to FSMs. We use BPMN as the notation for these BPMs and define our transformation accordingly.

Each task in BPMN is transformed to a corresponding FSM part. There are two states created for each task and a transition between them. Figure 4 shows the result of the transformation of the task *Create Invoice* of Figure 1.

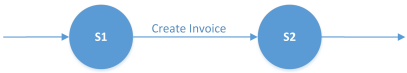


Figure 4: FSM Part for Task *Create Invoice*

S1 and *S2* represent the states before and after the execution of the *Create Invoice* task. The *hasEffect* predicate models the difference between the two states caused by this task execution, but not the states themselves.

The control flow between tasks in BPMN is directly translated to the FSM and connects the two adjacent tasks. For example, if the task *Transmit Invoice* from Figure 1 is transformed into an FSM, then it is connected to the previously transformed task *Create Invoice* according to the control flow defined in BPMN. The result of the connected tasks in the FSM is shown in Figure 5.

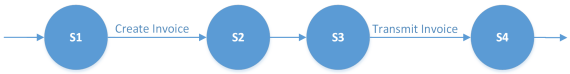


Figure 5: Connected FSM

Transitions between states in the FSM are executed immediately. Only the tasks that are transformed to FSMs may alter the state of the process. However, since only tasks may change the state of the process, control flow elements in BPMN can be optimized in the FSM. That is, they can be omitted in the resulting FSM. Figure 6 shows the resulting FSM for the example above.

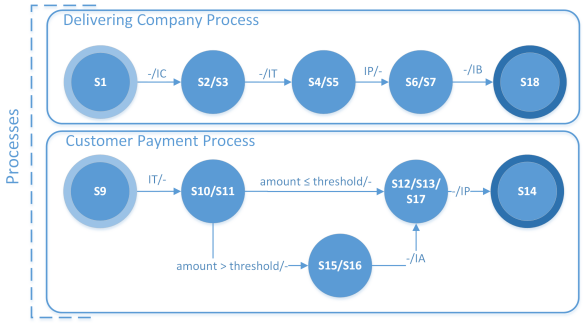
Figure 6: Optimized FSM

5.3 Setting Signals in FSMs Transferred from BPMN

FSMs use signals on state transitions to determine the next state. These signals are also used to trigger other states.

The FSM that we transfer from BPMN does not contain any signals yet. The outgoing signals of the states correspond to the effects of the tasks. Since each task may change the state of the process and these changes are defined via the *hasEffect* predicate in OWL-S, the signals can be derived from the effects. Each effect specification in OWL-S is in the form *predicate(Subject, Object)*. Hence, the signal can be derived as *SubjectObject*. The predicate can be omitted here, since in our case the predicate always relates to the *nextState* predicate of the object life cycle.

For example, the *Create Invoice* task has the effect *nextState(Invoice, Created)*. Hence, the corresponding signal is *InvoiceCreated*. The same applies for all other tasks. The resulting synchronized FSMs including the signals are shown in Figure 7.



IC = InvoiceCreated; IT = InvoiceTransmitted; IA = InvoiceAuthorized; IP = InvoicePaid; IB = InvoiceBooked.

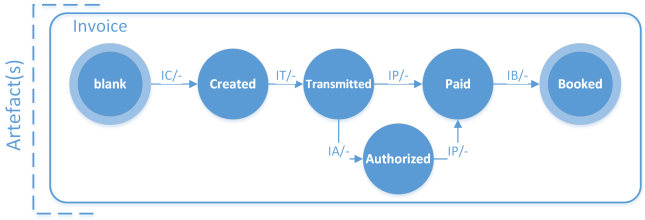
Figure 7: Resulting Synchronized FSMs

BPMN *Send-* and *Receive-Tasks* as shown in Figure 1 are a special kind of task and thus have to be treated separately. These tasks may not be implemented by a service, but are also specified using OWL-S. In case of the *Receive-Task*, a waiting operation for an incoming message is in place. An incoming message has to be received first, and only then the next transition is executed. Hence, an incoming signal is necessary in the FSM to model this waiting operation. The corresponding *Receive-Task*, for example *Receive Invoice*, has an incoming signal and is only triggered when this signal is set by another task. This signal is modeled in the task specification as a precondition of the form *stateOfInvoice(Invoice, State)*. In case of the *Receive Invoice* task, the incoming signal is *InvoiceTransmitted*. Hence, the resulting FSM uses signals that directly correspond to the effects of the tasks in BPMN as shown in Figure 1.

5.4 Connecting Process FSMs with Object Life Cycle FSMs

The only part missing before the verification approach defined in [7] can be used, is the connection of the FSMs transferred from the BPMN process with the object life cycle FSM. To accomplish this, we have to specify signals on the object life cycle FSM as well.

This can be done analogously to the way of defining signals described above. Since the object life cycle FSM does not change its states itself, an external signal is necessary. Each state in the object life cycle FSM can have a number of successor states. These states are identified via the *nextState* predicate. Again, the predicate can be omitted, and only the subject and the object are of interest. To be more precise, the subject, which is in this case the previous state, is not even necessary. For example, the state transition from *Created* to *Transmitted* is specified via the *nextState* predicate. It is of the form *nextState(Created, Transmitted)*. However, we need additional information to create the signal, since we have to identify the object that the transition is based upon. When *stateOfInvoice(Invoice, Created)* and *nextState(Created, Transmitted)* are known, according to a logical analysis of these two statements, the next state of the *Invoice* must be *Transmitted*. Thus we can derive *stateOfInvoice(Invoice, Transmitted)* and determine *InvoiceTransmitted* as the signal of the state *Created*. If a state has more than one successor states, then the corresponding signal has to be derived for each transition. This results in the FSM shown in Figure 8.



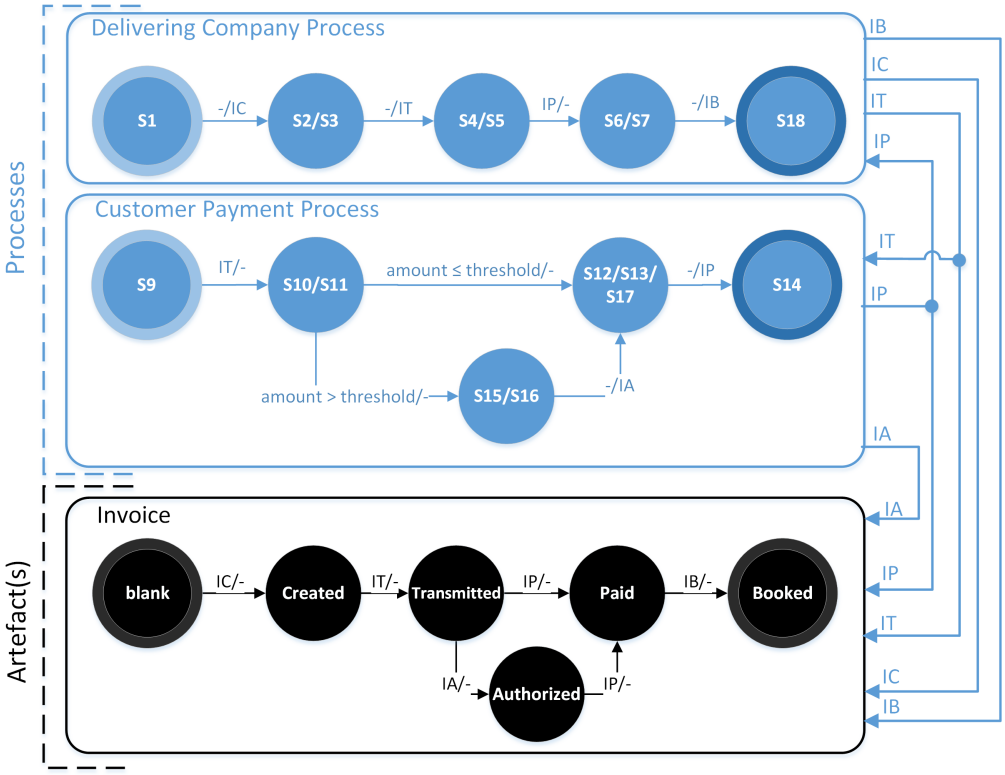
IC = InvoiceCreated; IT = InvoiceTransmitted; IA = InvoiceAuthorized; IP = InvoicePaid; IB = InvoiceBooked.

Figure 8: Object Life Cycle FSM with Signals

Since these signals correspond to the signals derived from the *hasEffect* predicate of the task specifications, the two FSMs for processes and object life cycles can be connected through them. This is illustrated in Figures 7 and 8 by the names of the signals. The resulting synchronized FSMs of both the processes and object life cycle are shown in Figure 9.

5.5 Using Synchronized FSMs for Formal Verification

This resulting machine is isomorphic to the one used in [7] for model checking. Therefore, this formal verification approach can be directly used with our systematic transformation for the generation of synchronized FSMs, while the



IC = InvoiceCreated; IT = InvoiceTransmitted; IA = InvoiceAuthorized; IP = InvoicePaid; IB = InvoiceBooked.

Figure 9: Connected Synchronized FSMs of BPMN Process and Object Life Cycle

approach for creating such synchronized FSMs in [7] was based on informal annotations.

Constraints on these business processes are defined as properties in a model checker language. These constraints impose certain restrictions on the enactment of the business process. A model checker uses these properties for formal verification and, in case of a violation, generates a counterexample.

In [7], these constraints are defined on the object life cycle, which decouples them from the BPM. That is, no complete knowledge of the BPM itself is necessary to define them. The advantage is, that these constraints can be reused on several BPMs and that a separation of concerns is achieved.

6 Discussion and Future Work

In artefact-centric modeling of business processes, artefacts and their life cycles are considered first-class citizens. This is in contrast to the more wide-spread modeling of business processes with the control flow between tasks in mind. Our approach tries to bridge the gap between these two philosophies by annotating the tasks in BPMs with semantic task specification, which is defined using the object life cycles of artefacts.

Since our new approach connects task- and artefact-centric models systematically, it may also be useful for automatically verifying their consistency. Such consistency checks are complementary to the approach by Lohmann et al. [8] as mentioned above, which employs compliance rules to automatically construct artefact-centric models from task-centric BPMs.

Actually, we have only connected a BPMN model with a single object life cycle yet. We plan to investigate the connection of several object life cycles and, if necessary, to extend our approach accordingly.

We tacitly assume that the OWL-S specification of the used Web services are defined using the same ontology as the one used for defining the object life cycles. In the context of an Enterprise Architecture, this assumption appears to be valid, because everything should be consistently defined within it.

In principle, the ideas of this approach would also allow the generation of Petri nets instead of synchronized FSMs. In essence, the signals would have to be mapped as additional tokens. In the context of this paper, we decided to generate synchronized FSMs, in order to use the model checking approach in [7].

SWRL-Rules can also be used as a means to express *con-*

straints to be satisfied during process execution. The formal verification approach presented in [7] represents constraints in the form of properties for model checking of business processes. These properties are formulated with respect to the life cycle of the artefacts, and they are directly encoded in the language used by the model checker. However, with our approach these constraints can also be defined in the semantic specification given in OWL and SWRL. McKenzie et al. [9] show how SWRL-Rules can be used to express fully quantified constraints, and we plan to use a similar approach to express constraints that have to be fulfilled during process execution. Listing 7 shows a constraint that states that each *Invoice* in the state *created* has to be *authorized* if the *amount* of the *Invoice* is *greater* than *5000*.

```
stateOfInvoice(?Invoice, created) ∧
    amountInvoice(?Invoice, ?amount) ∧
    swrlb:greaterThan(?amount, 5000) ⇒
    stateOfInvoice(?Invoice, authorize)
```

Listing 7: Constraint encoded as an SWRL-Rule

7 Conclusion

In this paper, we present a new and systematic approach for connecting BPMs (given in BPMN) with object life cycle models through semantic task specification. This specification is also the basis for connecting the behavioral models of the tasks and of their complementary object life cycles. More generally, this approach implements a connection between task- and artefact-centric models.

We also present the use of this connection for a seamless formal verification approach of BPMs through model checking. In contrast to the previous approach in [7] that we build upon, we use formal instead of informal annotations, and our new way of transforming BPMN tasks to synchronized finite state machines is consistent with the representation of object life cycles as used for model checking. Only through a formal approach as presented in this paper, the overall verification process can be made rigid.

8 Acknowledgment

Part of this research has been carried out in the ProREUSE project (No. 834167), funded by the Austrian FFG.

References

[1] OWL-based web service ontology version 1.2. http://www.daml.org/services/owl-s/1.2/, March 2004.

[2] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[3] M. Estañol, M.-R. Sancho, and E. Teniente. Verification and validation of UML artifact-centric business process models. In J. Zdravkovic, M. Kirikova, and P. Johannesson, editors, *Advanced Information Systems Engineering*, volume 9097 of *Lecture Notes in Computer Science*, pages 434–449. Springer International Publishing, 2015.

[4] Y. Feng and M. Kirchberg. Verifying OWL-S service process models. In *Proceedings of the 2011 IEEE International Conference on Web Services*, ICWS '11, pages 307–314, Washington, DC, USA, 2011. IEEE Computer Society.

[5] J. A. Fisteus, L. S. Fernández, and C. D. Kloos. Applying Model Checking to BPEL4WS Business Collaborations. In *Proceedings of the 2005 ACM Symposium on Applied Computing*, SAC '05, pages 826–830, New York, NY, USA, 2005. ACM.

[6] R. Hoch, H. Kaindl, R. Popp, D. Ertl, and H. Horacek. Semantic Service Specification for V&V of Service Composition and Business Processes. In *Proceedings of the 48nd Annual Hawaii International Conference on System Sciences (HICSS-48)*, Piscataway, NJ, USA, 2015. IEEE Computer Society Press.

[7] R. Hoch, M. Rathmair, H. Kaindl, and R. Popp. Verification of Business Processes Against Business Rules Using Object Life Cycles. In *New Advances in Information Systems and Technologies - Volume 1 [WorldCIST'16, Recife, Pernambuco, Brazil, March 22-24, 2016].*, volume 444 of *Advances in Intelligent Systems and Computing*, pages 589–598. Springer, 2016.

[8] N. Lohmann. Compliance by design for artifact-centric business processes. *Information Systems*, 38(4):606 – 618, 2013.

[9] C. McKenzie, P. Gray, and A. Preece. *Extending SWRL to Express Fully-Quantified Constraints*, pages 139–154. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[10] A. Meyer and M. Weske. Weak conformance between process models and synchronized object life cycles. In X. Franch, A. Ghose, G. Lewis, and S. Bhiri, editors, *Service-Oriented Computing*, volume 8831 of *Lecture Notes in Computer Science*, pages 359–367. Springer Berlin Heidelberg, 2014.

[11] Y. Ni and Y. Fan. Model transformation and formal verification for semantic web services composition. *Advances in Engineering Software*, 41(6):879–885, 2010.

[12] T. O. M. G. (OMG). Business Process Model and Notation (BPMN). http://www.omg.org/spec/BPMN/2.0, Jan. 2011. [Online; accessed 08-February-2015].

[13] Z. Sbai, A. Missaoui, K. Barkaoui, and R. Ben Ayed. On the verification of business processes by model checking techniques. In *Software Technology and Engineering (ICSTE), 2010 2nd International Conference on*, volume 1, pages V1–97–V1–103, Oct 2010.

[14] J. Schumacher and M. Meyer. *Customer Relationship Management strukturiert dargestellt: Prozesse, Systeme, Technologien*. Springer Berlin Heidelberg, 2003.

[15] W3C. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. https://www.w3.org/Submission/SWRL/, March 2004.

[16] I. Weber, J. Hoffmann, and J. Mendling. Beyond soundness: on the verification of semantic business process models. *Distributed and Parallel Databases*, 27(3):271–343, 2010.

[17] M. Wynn, H. Verbeek, W. van der Aalst, A. ter Hofstede, and D. Edmond. Business process verification – finally a reality! *Business Process Management Journal*, 15(1):74–92, 2009.